SECURITY CLASSIFICATION OF THIS PAGE

REPORT

AD-A236 200

DTIC ELECTE JUN 04 1991 S U

1a. REPORT SECURITY CLASSIFICATION

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

Approved for public release
Distribution Unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION: SUNY at Stony Brook

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION: Office of Naval Research

6c. ADDRESS (City, State and ZIP Code):
Computer Science Department
SUNY at Stony Brook
Stony Brook, NY 11794-4400

7b. ADDRESS (City, State and ZIP Code): Arlington, VA 22217-5500

8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Office of Naval Research

8b. OFFICE SYMBOL (If applicable)

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014-88-K-0383

8c. ADDRESS (City, State and ZIP Code): Arlington, VA 22217-5500

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE (Include Security Classification): Algorithms to Harness Massive Parallelism

12. PERSONAL AUTHOR(S): Larry D. Wittie

13a. TYPE OF REPORT: Final

13b. TIME COVERED: FROM 88/05/01 TO 89/09/30

14. DATE OF REPORT (Yr., Mo., Day): 16 October 89

15. PAGE COUNT: 7+100

16. SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB. GR

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): distributed control algorithms, error recovery, hierarchy reconfiguration, distributed shared memory, simulation of parallel programs, parallel debugging.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This research project has been a study of many ways to control and to use massive parallelism in distributed computing systems. This research emphasized the use of hierarchies to simplify control and application algorithms intended for parallel computer systems of thousands or millions of processors. Most of the algorithms considered were organized to use interprocessor message paths that formed virtual spanning trees touching every node in a network. Simulators were developed to predict the collective behavior of hundreds or thousands of computers running local copies of the same algorithm; a graphical interface (PARVU) was developed to display moving color images of parallel program behavior in large networks; and fiber-optic (MERLIN) hardware interfaces were designed to provide limited sharing of memory in networks of many hundreds computers running scientific application programs.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐

21. ABSTRACT SECURITY CLASSIFICATION: Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL: Larry D. Wittie

22b. TELEPHONE NUMBER (Include Area Code): (516) 632-8465

22c. OFFICE SYMBOL

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

Algorithms to Harness Massive Parallelism

Larry D. Wittie
Computer Science, SUNY, Stony Brook, NY 11794-4400
(516) 632-8456
lw@sbcs.sunysb.edu

1. Abstract

This research project has been a study of many ways to control and to use massive parallelism in distributed computing systems. This research emphasized the use of hierarchies to simplify control and application algorithms intended for parallel computer systems of thousands or millions of processors. Most of the algorithms considered were organized to use interprocessor message paths that formed virtual spanning trees touching every node in a network. Simulators were developed to predict the collective behavior of hundreds or thousands of computers running local copies of the same algorithm; a graphical interface (PARVU) was developed to display moving color images of parallel program behavior in large networks; and fiber-optic (MERLIN) hardware interfaces were designed to provide limited sharing of memory in networks of many hundreds computers running scientific application programs. Six reports and papers have thus far been written with partial support from this grant.

2. Technical Progress

Four lines of research were pursued under this grant:

(1) Analysis of control algorithms for distributed systems, especially three tree algorithms for restoration of control hierarchies after node failures;

(2) Simulation of hundreds of nodes running copies of the same distributed control algorithm, including a family of tree-cache algorithms that use a tree of name:location caches to speed searches for named resources;

(3) A graphical frontend for parallel system simulators that gives users animated color images of complex behavior to understand, debug, and evaluate distributed algorithms; and

(4) Analysis and preliminary design of hardware, distributed operating system, and programming support tools needed for Merlin shared-memory networks.

Because of the complex software systems involved, especially in the simulations, much of the research produced results suitable for publication only very late in the life of this short grant. Work is continuing to publish these results.

2.1. Analysis of Three Tree Recovery Algorithms

The new work has created a more careful explanation of analysis techniques to derive exact formulas for four performance criteria for three algorithms. The failure of a single node in a network computer managed by a control hierarchy requires local reconfiguration. A new logical tree must replace the older tree which lost a node. Logical neighbors of the failed node must detect the failure and rebuild the tree to maintain

network functionality. In the revised version of the paper "Hierarchy Reconfiguration Algorithms for Large Networks", three incremental tree reconfiguration algorithms are presented and their performances analyzed, evaluated and compared, especially for networks with thousands or millions of nodes. Performance is measured in terms of limits to disruption spread, path elongation, message traffic, and reconfiguration duration for each algorithm. Measures are determined algebraically for very large two-dimensional mesh-connected networks.

The simple *Leaf-Promotion* algorithm is recommended since analyses reveal that it performs best, at least for homogeneous networks. The recursive *Chaining* algorithm is best for heterogeneous networks in which lower level nodes are dedicated to peripheral functions or otherwise are less capable. This long paper has been submitted to a journal.

2.2. Simulation of Distributed Algorithms

Three progressively more efficient simulators have been implemented to study a family of distributed search algorithms that use a tree of fixed sized caches to find locations of named resources. Every network node keeps a cache of name:location pairs for all resources that have recently been accessed from the local node or from another node with a tree path to the resource that passes through the local node. Whenever a search is ended by finding a matching name:location entry in some cache, not only is that entry marked as being recently used, but the same entry is marked or inserted in every cache along the tree path from the requesting node to the location of the named resource.

Each cache should be the same size regardless of position in the tree. Otherwise, the algorithm would not be appropriate for arbitrarily large networks. Since cache size is limited, a least recently used (LRU) replacement algorithm is used to decide what old entry to remove before inserting any new one. Depending somewhat on the distribution of requesting (Q) node positions in the tree to the locations (L) of named resources (S), nodes near the root will tend to lie on the tree paths for many more Q to L references than will nodes nearer the leaves. Cache entries in near-root nodes change very rapidly.

The cache use and update part of the algorithm must be coupled with a policy on how a search of non-local caches is performed whenever a named resource is not found in the local cache. There are three tree search strategies, labeled slow, medium, and fast. The faster strategies have shorter average delays but unleash higher average message traffic onto the net.

The slow strategy recursively searches the local node first, then the branches of the tree below it, one at a time until some cache reports back to requester Q with the location L of the named resource S. If all lower branches fail to locate S, the search is passed up one tree level. To distribute the load of search messages uniformly across all branches at the same level, each search below should start at a different branch and then proceed in a cycle through all the branches.

The medium delay search strategy recursively tries the local node first, then starts simultaneous searches of all the direct subbranches. If one or more finds a S:L entry for the desired resource name, each reports back to the original requesting node Q. If all searches below a node fail, the search is passed up to the parent to search all branches except this one.

The fast strategy searches the network as fast as possible by checking the local cache first and if not found, passing the search simultaneously up to the parent and

down to all direct children. This fast search rapidly floods the tree with messages requesting the location L of resource S. In almost all cases, the message traffic generated by fast searches for all new resources is intolerably high.

Results show that fixed-size caches are acceptable if the expected number of nodes that access each leaf cache is bounded by locality constraints. The local distribution of requests used in these simulations was H (=tree height) sets of nodes with equal probability of being the location for a new request: self, all siblings, all first cousins, all second cousins, ... , and all remaining leaves. For this distribution, empirical results indicate that a fixed cache size of 32 or 64 nodes gives adequate speedup even in networks of hundreds of nodes. This result is comparable to the working set hypothesis for memory caches. Fixed-size caches do not work in large networks if resources are uniformly distributed. Upper-level caches, ie near the root, may not be useful in networks of hundreds of nodes, but more study is needed to say definitely.

The medium search strategy is a good compromise: it is nearly as fast as the flooding strategy and generates only slightly more traffic than the slow, sequential tree search. It remains to be seen if another cache policy is more effective than that of updating all caches on the path from source-where-named to destination-where-located for each requested resource. Work is continuing on simulation and analysis of this family of tree-based search algorithms.

2.3. Parvu: Parallel Views of Distributed Algorithms

Parvu is a visualization package for viewing the results of simulations of parallel systems as animated color images on Sun workstations. Parvu runs in C on a relatively slow Sun 3/260 (4 MIPS - millions of instructions per second) workstation with no hardware support for projections of three dimensional (3D) images. Parvu should run much faster on modern 3D workstations. The simulations can be run on any machine that can produce an ASCII file listing important behavioral events and the values of key state variables during simulation. The display can be produced concurrently with an on-going simulation or after it has completed. Different display representations of the events and states can be selected by the user during each run of the Parvu system on the same or different event lists.

At the moment, the Parvu system is limited to graphs of nodes and edges connected as trees. However, it is planned to extend it to cover other topologies including 2D and 3D meshes. It has been applied to explore mass behavior in a family of distributed search algorithms that use limited-size local caches to speed up hunts for the locations of named resources in networks of hundreds of computers. It can be applied to many other distributed algorithms, including network communication protocols, network traffic monitors, and Petri nets. Once extended to arbitrary topologies, it can be used to explore performance in many kinds of computer networks and to many classes of problems with discrete-element graph representations. The possible applications extend from computers linked by busses or communication lines to organic molecules with atoms linked by shared electrons.

The Parvu system has been used to view message traffic and cache hit ratios in large trees of computers running cache-assisted search algorithms. The basic tree image of up to 2,000 nodes is generated automatically by Parvu, but nodes and subtrees can be repositioned at will by the user. Significant events may be flagged by flashing color changes for specific nodes and links For these algorithms, important events include:

arrival at a randomly selected leaf of a new request to find the location of an object known only by name, search of the local name:location cache, sending of a message to pass the search to another node, and sending of a backward propagating reply message that tells the initial requester where the named object is located and that updates caches along its return path.

New nodes flash red in the animated image as they are searched. Once a resource location is determined either from the location node itself or from an earlier cache, a visible token is passed along the path from the location to the node that first requested the search. Numbers in the nodes give cache hit ratios for requests from above and from below in the tree. Mathematical analyses of the tree-cache algorithms indicate very different dynamics for requests: those from above are essentially random and those from below reflect any locality in the request patterns.

The PARVU animation system has proven very handy for detecting errors in the coding of the cache-update and search routines. An early implementation of the tree-cache algorithm always started slow sequential searches of subtrees with the leftmost child, resulting in heavier traffic on the left of the tree than on the right. A display of average branch traffic showed the misbalance. The slow search algorithm was modified to choose each first subtree at random.

For the medium speed search, a bug in the code sometimes caused searching to continue from the root after a resource had been found in a lower cache. Because of this error, caches above the node finding the name were also queried and searching was resumed if the name was not found. The mistake occurred often at the root since cache entries near the root are rapidly flushed by the high activity there. The faulty termination of some searches showed clearly in the Parvu views of tree-cache behavior, long before it made a noticeable affect on cache hit ratios. Parvu has made feasible the exploration of a large family of related search algorithms.

2.4. Merlin: Network Designs for Massively Parallel Computation

Software and hardware for a new kind of massively parallel computing system are being developed at Stony Brook and Sandia. The Merlin project is presently planning to link 8 to 64 fast computers to provide more than 1,000 MIPS of computing power and to refine a mapped, reflective memory technique for joining many types of computers to form heterogeneous supercomputers. Special interfaces let processors selectively share data on a word-by-word basis with low latency. Fast firmware virtual circuits are reconfigured to match topological needs of individual application programs. The Merlin prototypes will be used to run scientific programs and to design a Teraflops supercomputer built from thousands of commercial computers.

Network traffic analyses show that the first Merlin interfaces, which pass more than 480 MegaBytes/second of data per processor, are fast enough to link hundreds of processors. Latency times for sharing single words randomly among 64 processors are only 2.1 microseconds with four pairs of 2.4 gigabit/second (Gb/s) optical fibers per node. With eight 24 Gb/s outputs, they can span 4,096 computers in only 0.4 microseconds.

The Merlin system (MEmory Routed, Logical Interconnection Network) uses high bandwidth (300 MByte/sec) communication links to provide an open architecture for tightly-coupled heterogeneous networks, especially for scientific applications. Words written into local memories are rapidly mirrored into selected remote memories via

programmable routing maps. The overall goal of the Merlin project is to produce a new computer architecture that can join fast computers to form heterogeneous supercomputers: a supercomputer glue.

Application computer hosts are linked by sets of fast optical fibers that provide the programmer with the illusion and convenience of globally shared memory without the long latency delays inherent to remote memories shared on demand and without the contention problems of any widely shared single memory. All memory modules are actually local to one or another of the processors. Global virtual memory does not exist; each global page is a spanning tree of buffers and routers to pass words among shared local pages. All data are shared via fast interfaces in the host backplanes without any host processor involvement after initialization. Most simulations of physical systems have massive amounts of data parallelism, many values that can be calculated simultaneously because they are in distant regions that rarely interact. Merlin hardware and software abstract away many of the architectural details normally needed to map parallel algorithms efficiently onto networks. Planned software tools will help users debug and improve their parallel application programs running on Merlin. Two 8 to 64 node Merlin prototypes are planned to be built in the next three years: 8 nodes providing about 320 million instructions per second (MIPS) at Stony Brook to develop software and probably 32 nodes (1,000+ MIPS) at Sandia. Each will run one large C or Fortran program at a time.

Merlin networking technology allows simplifies parallel debugging on a wide variety of computers, eases the parallel programming problem by giving the illusion of shared global memory, and encourages the use of very fast specialized hardware in massively parallel heterogeneous networks. The Merlin prototypes will be used to develop operating system and debugging software for running large scientific application programs and to improve interconnection hardware, operating systems, and programming support tools that will allow the creation of a 4,096 node, 1 to 2 Teraflops multi-user computing system within six years.

3. Software and Hardware Prototypes

Parvu is software of general use that has been produced as part of this research. Parvu is a visualization package for viewing the results of simulations of parallel systems as animated color images on Sun workstations. The simulations can be run on any machine that can produce an ASCII file listing important behavioral events and the values of key state variables during simulation. The display can be produced concurrently with an on-going simulation or after it has completed. Different display representations of the events and states can be selected by the user during each run of the Parvu system on the same or different event lists.

At the moment, the Parvu system is limited to graphs of nodes and edges connected as trees. However, it is planned to extend it to cover other topologies including 2D and 3D meshes. It has been applied to explore mass behavior in a family of distributed search algorithms that use limited-size local caches to speed up hunts for the locations of named resources in networks of hundreds of computers. It can be applied to many other distributed algorithms, including network communication protocols, network traffic monitors, and Petri nets. Once extended to arbitrary topologies, it can be

used to explore performance in many kinds of computer networks and to many classes of problems with discrete-element graph representations. The possible applications extend from computers linked by busses or communication lines to organic molecules with atoms linked by shared electrons. Code for the current version of Parvu works well.

Plans for efficient hardware implementations to support program synchronizations in Merlin networks are nearly complete. Construction of working prototypes of the Merlin interfaces for low-latency sharing of pre-selected memory regions in large computer networks awaits substantial funding. Several reports and three full papers on Merlin have already been produced with partial support from the Office of Naval Research.

4. Summary of Technical Progress

Over the course of this research into massively parallel algorithms, work has progressed from mathematical analyses of complex distributed algorithms, to more practical simulations with visual frontends for rapid assessment of parallel system behavior, to the design of generally applicable techniques for linking computer memories. Perhaps not surprisingly, the hardest theoretic issues have arisen in developing fast, but sufficient algorithms for control of large shared-memory networks.

5. Lists of Publications, Presentations, and Reports

The six starred(*) papers and technical reports are appended to this report.

5.1. Refereed papers published

*1) Larry D. Wittie and Creve Maples, "Merlin: Massively Parallel Heterogeneous Computing", *Proc. 1989 Int. Conf. on Parallel Processing,* St. Charles, Illinois, August 1989, pp. 142-150.

*2) Larry D. Wittie, "Debugging Distributed C Programs by Real Time Replay", *Proc. ACM Workshop on Parallel and Distributed Debugging,* Madison, WI, May 1988, pp. 57-67.

5.2. Refereed papers submitted

*1) Creve Maples and Larry Wittie, "Merlin: A Superglue for Multicomputer Systems", Submitted for publication, Sept. 1989.

2) Larry Wittie, Johannes Sayre, and Creve Maples, "Mirror Memory for Massively Parallel Scientific Programs", Submitted for publication, Sept. 1989. (Revision of CFD paper)

*3) Larry Wittie and C.K. Mohan, "Hierarchy Reconfiguration Algorithms for Large Network Computers", Submitted for publication, July 1989

5.3. Unrefereed reports and articles

*1) Larry Wittie, Johannes Sayre, and Creve Maples, "Massively Parallel Memory Sharing for CFD Codes", Parallel CFD Conference, Los Angeles, CA, May 8, 1989.

*2) Larry D. Wittie and Creve Maples, "Network Traffic for Massively Parallel Memory Sharing", SUNY/Stony Brook, Comp. Sci. Tech. Rep. 89/06, Jan. 1989, 13 pp.

3) Larry D. Wittie and Creve Maples, "Merlin: Massively Parallel Heterogeneous Computing", SUNY/Stony Brook, Computer Science Tech. Rep. 88/21, Dec. 1988, 39 pp.

5.4. Invited presentations

1) Larry D. Wittie, "Merlin: Shared Memory Distributed Systems", Invited panelist for 9th Int. Conf. on Distributed Computing Systems, Newport, CA, June 7, 1989.

2) Larry D. Wittie, "Through the MERLIN Looking Glass: Running Large Parallel Scientific Programs", NSF, Washington, DC, Nov. 7, 1988.

3) Larry D. Wittie,"Software Support for Merlin", National Security Agency, Elk Ridge, MD, Oct. 18, 1988.

4) Larry D. Wittie,"Software Support for Merlin", Supercomputer Research Center, Lanham, MD, Oct. 19, 1988.

5.5. Contributed presentations

1) Larry D. Wittie and Creve Maples, "Merlin: Massively Parallel Heterogeneous Computing", *Proc. 1989 Int. Conf. on Parallel Processing,* St. Charles, Illinois, August 1989, pp. 142-150.

2) Larry Wittie, Johannes Sayre, and Creve Maples, "Massively Parallel Memory Sharing for CFD Codes", Parallel CFD Conference, Los Angeles, CA, May 8, 1989.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT

AD-A236 200

DTIC ELECTE JUN 04 1991 S U

1a. REPORT SECURITY CLASSIFICATION

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
STATEMENT A
Approved for public release;
Distribution Unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
SUNY at Stony Brook		Office of Naval Research

6c. ADDRESS (City, State and ZIP Code)	7b. ADDRESS (City, State and ZIP Code)
Computer Science Department SUNY at Stony Brook Stony Brook, NY 11794-4400	Arlington, VA 22217-5500

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
Office of Naval Research		N00014-88-K-0383

8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS.			
	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
Arlington, VA 22217-5500				

11. TITLE (Include Security Classification)
Algorithms to Harness Massive Parallelism

12. PERSONAL AUTHOR(S)
Larry D. Wittie

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM 88/05/01 TO 89/09/30	16 October 89	7+100

16. SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
distributed control algorithms, error recovery, hierarchy reconfiguration, distributed shared memory, simulation of parallel programs, parallel debugging.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This research project has been a study of many ways to control and to use massive parallelism in distributed computing systems. This research emphasized the use of hierarchies to simplify control and application algorithms intended for parallel computer systems of thousands or millions of processors. Most of the algorithms considered were organized to use interprocessor message paths that formed virtual spanning trees touching every node in a network. Simulators were developed to predict the collective behavior of hundreds or thousands of computers running local copies of the same algorithm; a graphical interface (PARVU) was developed to display moving color images of parallel program behavior in large networks; and fiber-optic (MERLIN) hardware interfaces were designed to provide limited sharing of memory in networks of many hundreds computers running scientific application programs.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Larry D. Wittie	(516) 632-8465	

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE